A Appendix: Code Listings

Listing 1: Board.hpp

```
#ifndef BOARD_H
#define BOARD_H

#include "Tile.hpp"
#include "Player.hpp"
#include "GroupsManager.hpp"
#include "CardsManager.hpp"
#include <vector>

class Board
{
public:
    //-------------------------------------------------------------------------
    /// @brief default contstructor
    //-------------------------------------------------------------------------
    Board();
    //-------------------------------------------------------------------------
    /// @brief method that prints the board
    //-------------------------------------------------------------------------
    void print(){
        for(unsigned int i=0; i<m_tiles.size();++i)
        {
            m_tiles[i]->print();
        }
    }
    //-------------------------------------------------------------------------
    /// @brief method that prints a specific tile
    /// @param[in] i_tile tile to be printed
    //-------------------------------------------------------------------------
    void print(unsigned int i_tile)const;
    //-------------------------------------------------------------------------
    /// @brief method that resets the board
    //-------------------------------------------------------------------------
    void reset();
    //-------------------------------------------------------------------------
    /// @brief method that perfoms an action
    //-------------------------------------------------------------------------
    void action( PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief method that allows the current Player to buy houses
    //-------------------------------------------------------------------------
    void buildHouses(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief method that returns the name of the tile
    //-------------------------------------------------------------------------
    const std::string &getTileName(unsigned int i_index)const;
    //-------------------------------------------------------------------------
    /// default destructor
    //-------------------------------------------------------------------------
    ~Board();


private:
    //-------------------------------------------------------------------------
    /// @brief how the tiles relates to each other, separated in groups
    //-------------------------------------------------------------------------
    GroupsManager m_groups;
    //-------------------------------------------------------------------------
    /// @brief method that read the name of the next tile
    //-------------------------------------------------------------------------
    std::string readTilesName(
            const std::vector<std::string> &i_words,
            unsigned int *i_p
            );
    //-------------------------------------------------------------------------
    /// @brief the number of tiles
```

```
    //--------------------------------------------------------------------------
    static const unsigned int numOfTiles = 40;
    //--------------------------------------------------------------------------
    /// @brief the cards, community chest and chance
    //--------------------------------------------------------------------------
    CardsManager m_cards;
    //--------------------------------------------------------------------------
    /// @brief all the tiles of the board in the correct order
    //--------------------------------------------------------------------------
    std::vector<Tile *> m_tiles;
};

#endif // BOARD_H
```

Listing 2: Board.cpp

```
#include <iostream>
#include <fstream>
#include <iterator>
#include <stdlib.h>

#include "Board.hpp"
#include "Order.hpp"
#include "Property.hpp"
#include "Station.hpp"
#include "Player.hpp"
#include "NormalProperty.hpp"
#include "Utility.hpp"
#include "GetACardTIle.hpp"

//--------------------------------------------------------------------------
Board::Board()
{
  //  CardsManager::initialiseCards();
    // Corners
    std::ifstream bvhStream("Board");
    if (!bvhStream) {
        std::cout << "File Board not found. Game will terminate\n";
        exit (EXIT_FAILURE);
    }
    std::istream_iterator<std::string> bvhIt(bvhStream);
    std::istream_iterator<std::string> sentinel;
    std::vector<std::string> words(bvhIt,sentinel);
    m_tiles.resize(40);
    unsigned int i=0 , counter = 0;
    while(i<words.size()-3)
    {
        i++;
        const char flag = words[i++][0];
        unsigned int price = 0;
        std::string name, colour;
        int housePrice;
        std::vector<unsigned int> rentPrices;
        rentPrices.resize(6);
        switch(flag)
        {
        case 'c': //Card
            name  = readTilesName(words,&i);
            m_tiles[counter] = new GetACardTIle(name,m_cards);
            break;
        case 'o': //Order tax,jail,go,go to jail
        {
            const std::string flag = words[i++];
            if(flag=="p")
            {
                const unsigned int money =atoi(words[i++].c_str());
                name = readTilesName(words,&i);
                m_tiles[counter]=new Order(name,"p",money);
            }
            else
```

```
            {
                name = readTilesName(words,&i);
                m_tiles[counter] = new Order(name,flag);
            }
            break;
        }
        case 'p': // Property
        {
            const char secondFlag = words[i++][0];
            price = atoi(words[i++].c_str());
            switch(secondFlag)
            {
            case 'n': //Normal Property
            {
                housePrice = atoi(words[i++].c_str());
                for(unsigned int k=0; k<6; ++k)
                {
                    rentPrices[k] = atoi(words[i++].c_str());
                }
                colour = words[i++];
                name = readTilesName(words,&i);
                m_tiles[counter] =
                        new NormalProperty(name,price,housePrice,rentPrices);
                m_groups.addTile(colour,m_tiles[counter]);
                m_tiles[counter]->setColour(colour);
                break;
            }
            case 's': //Station
                for(unsigned int k=0; k<4; ++k)
                {
                    rentPrices[k] = atoi(words[i++].c_str());
                }
                name = readTilesName(words,&i);
                m_tiles[counter] = new Station(name,price,rentPrices);
                m_groups.addTile("station",m_tiles[counter]);
                break;
            case 'u' : // Works - Company
                for(unsigned int k=0; k<2; ++k)
                {
                    rentPrices[k] = atoi(words[i++].c_str());
                }
                name = readTilesName(words,&i);
                m_tiles[counter] = new Utility(name,price,rentPrices);
                m_groups.addTile("utility",m_tiles[counter]);
            break;
            default:
                break;
            }
            break;
        }
        default:
            std::cerr << "Wrong flag type.\n";
            exit(EXIT_FAILURE);
            break;
        }
        i++;
        counter++;
    }
}

//--------------------------------------------------------------------------
void Board::print(unsigned int i_tile) const
{
    m_tiles[i_tile]->print();
}

//--------------------------------------------------------------------------
void Board::action(PlayerManager &i_players)
{
    int currentTile = i_players.getPosition();
    m_tiles[currentTile]->action(i_players);
```

```
}

//------------------------------------------------------------------------
void Board::reset()
{
    for(unsigned int i=0; i<numOfTiles; ++i)
    {
        m_tiles[i]->reset();
    }
}


//------------------------------------------------------------------------
const std::string &Board::getTileName(unsigned int i_index) const
{
    return m_tiles[i_index]->getName();
}



//-------------------------------------------------------------------------------
std::string Board::readTilesName(
        const std::vector<std::string> &i_words, unsigned int *io_p
        )
{
    std::string name = i_words[*io_p];
    *io_p = *io_p + 1;
    while(i_words[*io_p]!=".")
    {
        name+=" " + i_words[*io_p];
        *io_p = *io_p + 1;
    }
    return name;
}

//------------------------------------------------------------------------
void Board::buildHouses(PlayerManager &i_players)
{
    m_groups.buildHouses(i_players);
}

//------------------------------------------------------------------------
Board::~Board()
{
   // CardsManager::destroyedAllCards();
    for (unsigned int i=0; i<numOfTiles; ++i)
    {
        delete m_tiles[i];
    }
}
```

Listing 3: CardChanceOrLoseMoney.hpp

```
#ifndef CARDCHANCEORLOSEMONEY_H
#define CARDCHANCEORLOSEMONEY_H

#include "Card.hpp"

class CardChanceOrLoseMoney :public Card
{
public:
    //------------------------------------------------------------------------
    /// @brief default constructor
    //------------------------------------------------------------------------
    CardChanceOrLoseMoney(unsigned int i_balance);
    //------------------------------------------------------------------------
    /// @brief method that performs the action associated with that card
    //------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //------------------------------------------------------------------------
```

```
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~CardChanceOrLoseMoney();

private:
    //-------------------------------------------------------------------------
    /// @brief the new position of the player
    //-------------------------------------------------------------------------
    int m_balanceToRemove;
};

#endif // CARDCARDCHANCEORLOSEMONEY_H
```

Listing 4: CardChanceOrLoseMoney.cpp

```
#include "CardChanceOrLoseMoney.hpp"


//-------------------------------------------------------------------------
CardChanceOrLoseMoney::CardChanceOrLoseMoney(
        unsigned int i_balance
        ):m_balanceToRemove(i_balance)
{
}

//-------------------------------------------------------------------------
void CardChanceOrLoseMoney::action(
        PlayerManager &i_players
        )
{
    std::cout << "Would you like to take a chance (Enter 'y') or Pay   " <<
                 m_balanceToRemove << " (Enter 'n'): " << std::endl;
    std::string decision = "";
    std::cin >> decision;

    if (decision == "y")
    {
      std::cout << "[Call Chance]" << std::endl;

    }
    else if (decision == "n")
    {
        if(i_players.takeBalance(m_balanceToRemove))
        {
            std::cout<< i_players.getName() << " paid " << m_balanceToRemove << std::
                endl;
        }
        else
        {
            std::cout << i_players.getName()
                      << " does not have enough money to pay" << std::endl;
        }

    }
    else
      std::cout << "Invalid Input. Please enter y or n" << std::endl;
}

//-------------------------------------------------------------------------
CardChanceOrLoseMoney::~CardChanceOrLoseMoney()
{
}
```

Listing 5: CardGetOutOfJail.hpp

```
#ifndef CARDGETOUTOFJAIL_H
#define CARDGETOUTOFJAIL_H

#include "Card.hpp"
```

```
class CardGetOutOfJail :public Card
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    CardGetOutOfJail();
    //-------------------------------------------------------------------------
    /// @brief method that performs the action associated with that card
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~CardGetOutOfJail();


};

#endif // CARDGETOUTOFJAIL_H
```

Listing 6: CardGetOutOfJail.cpp

```
#include "CardGetOutOfJail.hpp"

//-------------------------------------------------------------------------
CardGetOutOfJail::CardGetOutOfJail()
{
}

//-------------------------------------------------------------------------
void CardGetOutOfJail::action(PlayerManager &i_players)
{
    i_players.setJailed(false);
    std::cout<< i_players.getName() << " Gets a Get Out of Jail Card " << std::endl;
}

//-------------------------------------------------------------------------
CardGetOutOfJail::~CardGetOutOfJail()
{
}
```

Listing 7: CardGoToJail.hpp

```
#ifndef CARDGOTOJAIL_H
#define CARDGOTOJAIL_H

#include "Card.hpp"

class CardGoToJail :public Card
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    CardGoToJail();
    //-------------------------------------------------------------------------
    /// @brief method that performs the action associated with that card
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~CardGoToJail();


};
```

```
#endif // CARDGOTOJAIL_H
```

Listing 8: CardGoToJail.cpp

```
#include "CardGoToJail.hpp"

//-------------------------------------------------------------------------
CardGoToJail::CardGoToJail()
{
}

//-------------------------------------------------------------------------
void CardGoToJail::action(
        PlayerManager &i_players
        )
{
  i_players.setJailed(true);
  std::cout<< i_players.getName() << " IS GOING TO JAIL!!" << std::endl;
}

//-------------------------------------------------------------------------
CardGoToJail::~CardGoToJail()
{
}
```

Listing 9: Card.hpp

```
#ifndef CARDTYPE_H
#define CARDTYPE_H
#include <iostream>
#include <vector>
#include "PlayerManager.hpp"

class Card
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    Card();
    //-------------------------------------------------------------------------
    /// @brief method that performs the action associated with that card
    //-------------------------------------------------------------------------
    virtual void action(PlayerManager &i_players)=0;
    //-------------------------------------------------------------------------
    /// @brief method that prints the card
    //-------------------------------------------------------------------------
    void print()const;
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    virtual ~Card();

};

#endif // CARDTYPE_H
```

Listing 10: Card.cpp

```
#include "Card.hpp"

//-------------------------------------------------------------------------
Card::Card()
{
}

//-------------------------------------------------------------------------
```

```
Card::~Card()
{

}
```

Listing 11: CardMovePlayerBack.hpp

```
#ifndef CARDMOVEPLAYERBACK_H
#define CARDMOVEPLAYERBACK_H

#include "Card.hpp"

class CardMovePlayerBack :public Card
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constractor
    //-------------------------------------------------------------------------
    CardMovePlayerBack(unsigned int i_position);
    //-------------------------------------------------------------------------
    /// @brief method that performs the action associated with that card
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~CardMovePlayerBack();

private:
    //-------------------------------------------------------------------------
    /// @brief the new position of the player
    //-------------------------------------------------------------------------
    int m_placesToMove;
};

#endif // CARDMOVEPLAYERTOBACK_H
```

Listing 12: CardMovePlayerBack.cpp

```
#include "CardMovePlayerBack.hpp"

//-----------------------------------------------------------------------------
CardMovePlayerBack::CardMovePlayerBack(
        unsigned int i_position
        ):m_placesToMove(i_position)
{
}

//-----------------------------------------------------------------------------
void CardMovePlayerBack::action(
        PlayerManager &i_players
        )
{
  i_players.movePositionBy(-m_placesToMove);
  std::cout<< "Move " << i_players.getName() << " back " << m_placesToMove << " spaces."
       << std::endl;
}

//-----------------------------------------------------------------------------
CardMovePlayerBack::~CardMovePlayerBack()
{
}
```

Listing 13: CardMovePlayerToPosition.hpp

```
#ifndef CARDMOVEPLAYERTOPOSITION_H
#define CARDMOVEPLAYERTOPOSITION_H
```

```
#include "Card.hpp"

class CardMovePlayerToPosition :public Card
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constractor
    //-------------------------------------------------------------------------
    CardMovePlayerToPosition(unsigned int i_position);
    //-------------------------------------------------------------------------
    /// @brief method that performs the action associated with that card
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~CardMovePlayerToPosition();

private:
    //-------------------------------------------------------------------------
    /// @brief the new position of the player
    //-------------------------------------------------------------------------
    int m_newPosition;
};

#endif // CARDMOVEPLAYERTOPOSITION_H
```

Listing 14: CardMovePlayerToPosition.cpp

```
#include "CardMovePlayerToPosition.hpp"

//-----------------------------------------------------------------------------
CardMovePlayerToPosition::CardMovePlayerToPosition(
        unsigned int i_position
        ):m_newPosition(i_position)
{
}

//-----------------------------------------------------------------------------
void CardMovePlayerToPosition::action(
        PlayerManager &i_players
        )
{
    i_players.setPosition(m_newPosition);
    std::cout<< "Move " << i_players.getName() << " to " << i_players.getPosition() << "
         position." << std::endl;
}

//-----------------------------------------------------------------------------
CardMovePlayerToPosition::~CardMovePlayerToPosition()
{
}
```

Listing 15: CardReceiveMoneyFromPlayers.hpp

```
#ifndef CARDRECEIVEMONEYFROMPLAYERS_H
#define CARDRECEIVEMONEYFROMPLAYERS_H

#include "Card.hpp"

class CardReceiveMoneyFromPlayers :public Card
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    CardReceiveMoneyFromPlayers(unsigned int i_balance);
    //-------------------------------------------------------------------------
```

```
    /// @brief method that performs the action associated with that card
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~CardReceiveMoneyFromPlayers();

private:
    //-------------------------------------------------------------------------
    /// @brief the new position of the player
    //-------------------------------------------------------------------------
    int m_balanceToAdd;
};

#endif // CARDRECEIVEMONEYFROMPLAYERS_H
```

Listing 16: CardReceiveMoneyFromPlayers.cpp

```
#include "CardReceiveMoneyFromPlayers.hpp"

//-------------------------------------------------------------------------
CardReceiveMoneyFromPlayers::CardReceiveMoneyFromPlayers(
        unsigned int i_balance
        ):m_balanceToAdd(i_balance)
{
}

//-------------------------------------------------------------------------
void CardReceiveMoneyFromPlayers::action(
        PlayerManager &i_players
        )
{
  //Uses method in PlayerManager
    i_players.takeBalance(i_players.getMoneyFromEachPlayer(m_balanceToAdd));
}

//-------------------------------------------------------------------------
CardReceiveMoneyFromPlayers::~CardReceiveMoneyFromPlayers()
{
}
```

Listing 17: CardReceiveMoney.hpp

```
#ifndef CARDRECEIVEMONEY_H
#define CARDRECEIVEMONEY_H

#include "Card.hpp"

class CardReceiveMoney :public Card
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    CardReceiveMoney(unsigned int i_balance);
    //-------------------------------------------------------------------------
    /// @brief method that performs the action associated with that card
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~CardReceiveMoney();

private:
    //-------------------------------------------------------------------------
    /// @brief the new position of the player
    //-------------------------------------------------------------------------
```

```
    int m_balanceToAdd;
};

#endif // CARDRECEIVEMONEY_H
```

Listing 18: CardReceiveMoney.cpp

```
#include "CardReceiveMoney.hpp"

//-------------------------------------------------------------------------
CardReceiveMoney::CardReceiveMoney(
        unsigned int i_balance
        ):m_balanceToAdd(i_balance)
{
}

//-------------------------------------------------------------------------
void CardReceiveMoney::action(
        PlayerManager &i_players
        )
{
    i_players.addBalance(m_balanceToAdd);
    std::cout << "Player " << i_players.getName() << " gets " << m_balanceToAdd << "\n";
}

//-------------------------------------------------------------------------
CardReceiveMoney::~CardReceiveMoney()
{
}
```

Listing 19: CardsManager.hpp

```
#ifndef CARDMANAGER_H
#define CARDMANAGER_H

#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <ctime>
#include "Player.hpp"
#include "Tile.hpp"
#include "Card.hpp"


class CardsManager
{
public:
    CardsManager();
    //-------------------------------------------------------------------------
    /// @brief method that does the action =p
    /// @param[in] i_players the players of the game
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players, const std::string &i_name);
    //-------------------------------------------------------------------------
    /// @brief method that prints all the information about the tile
    //-------------------------------------------------------------------------
    void print()const;

    void printAllCards()
    {
        for(unsigned int i=0; i<m_communityChest.size(); ++i)
        {
            m_communityChest[i]->print();
        }
    }
    //-------------------------------------------------------------------------
    /// @brief default destructor
```

```
    //--------------------------------------------------------------------------
    virtual ~CardsManager();

private:

    //--------------------------------------------------------------------------
    /// @brief method that initialises all the cards
    /// @brief it MUST BE CALLED before this class is used
    //--------------------------------------------------------------------------
    void initialiseCards();

    //--------------------------------------------------------------------------
    /// @brief all the community Chest cards
    //--------------------------------------------------------------------------
    std::vector<Card *> m_communityChest;
    //--------------------------------------------------------------------------
    /// @brief all the chance cards
    //--------------------------------------------------------------------------
    std::vector<Card *> m_chance;

};


#endif //CardMAnager
```

Listing 20: CardsManager.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <ctime>

#include "Player.hpp"
#include "CardsManager.hpp"
#include "CardMovePlayerToPosition.hpp"
#include "CardStreetRepairs.hpp"
#include "CardReceiveMoney.hpp"
#include "CardChanceOrLoseMoney.hpp"
#include "CardGoToJail.hpp"
#include "CardGetOutOfJail.hpp"
#include "CardTakeMoney.hpp"
#include "CardMovePlayerBack.hpp"
#include "CardReceiveMoneyFromPlayers.hpp"

//Vectors to hold String from text files containing Card Specifications
std::vector<std::string> communityChestCards;
std::vector<std::string> chanceCards;

CardsManager::CardsManager()
{
    initialiseCards();
}

void CardsManager::action(PlayerManager &i_players, const std::string &i_name)
{
  //Uses Tile Name to decided what action to take
  //Picks random card from 'deck'
    if(i_name=="CHANCE")
    {
        const unsigned int random = rand() % m_chance.size();
        m_chance[random]->action(i_players);
        std::cout << chanceCards[random] << std::endl;
    }
    else if(i_name == "COMMUNITY CHEST")
    {
        unsigned int random = std::rand() % m_communityChest.size();
        m_communityChest[random]->action(i_players);
        std::cout << communityChestCards[random] << std::endl;
```

```
    }
    else
    {
        std::cout << "This is not a card type!" << std::endl;
    }
}

void CardsManager::initialiseCards()
{
  //Read Card specifications from file
    std::string line;
    std::ifstream myfile1 ("CommunityChest");
    if (myfile1.is_open())
    {
        getline(myfile1, line);
        do
        {
            communityChestCards.push_back(line);
            getline(myfile1, line);
        }
        while (!myfile1.eof());
    }
    myfile1.close();

    std::ifstream myfile2 ("Chance");
    if (myfile2.is_open())
    {
        getline(myfile1, line);
        do
        {
            chanceCards.push_back(line);
            getline(myfile2, line);
        }
        while (!myfile2.eof());

    }
    myfile2.close();


  //Parse Each string in Comminity Chest vector and organise them into appropriate Cards
  //Store each newly created Card into Community Chest Card Vector
    m_communityChest.resize(communityChestCards.size());
    for(unsigned int i=0; i<communityChestCards.size(); ++i)
    {

        line = communityChestCards[i];
        std::istringstream iss(line);
        std::string sub;

        //First Sub-String of line is always flag
        iss >> sub;

        if (sub == "b") //Binary choice (e.g pay  1 0  or take a chance)
        {
            int moneyToRemove;
            iss >> sub;
            std::istringstream(sub) >> moneyToRemove;
            m_communityChest[i]= new CardChanceOrLoseMoney(moneyToRemove);
        }

        else if (sub == "f") //Get out of Jail Free
        {
          m_communityChest[i] = new CardGetOutOfJail();
        }

        else if (sub == "g") //Player RECIEVES money
        {
            int moneyToAdd;
            iss >> sub;
            std::istringstream(sub) >> moneyToAdd;
            m_communityChest[i] = new CardReceiveMoney(moneyToAdd);
```

```
        }

        else if (sub == "gp") //Receive money from EACH player
        {
            int moneyToAdd;
            iss >> sub;
            std::istringstream(sub) >> moneyToAdd;
            m_communityChest[i] = new CardReceiveMoneyFromPlayers(moneyToAdd);
        }

        else if (sub == "h") //Player has to perform repairs on property
        {
            int house, hotel;
            iss >> sub;
            std::istringstream(sub) >> house;
            iss >> sub;
            std::istringstream(sub) >> hotel;
            m_communityChest[i] = new CardStreetRepairs(house, hotel);
        }

        else if (sub == "j") //Go To Jail
        {
          m_communityChest[i] = new CardGoToJail();
        }

        else if (sub == "l") //Take money from player
        {
            int moneyToRemove;
            iss >> sub;
            std::istringstream(sub) >> moneyToRemove;
            m_communityChest[i] = new CardTakeMoney(moneyToRemove);
        }

        else if (sub == "m") //Move player TO SPECIFIED position
        {
            int boardPosition;
            iss >> sub;
            std::istringstream(sub) >> boardPosition;
            m_communityChest[i] = new CardMovePlayerToPosition(boardPosition);
        }

        else if (sub == "mb") //Move player back certain amount of spaces
                              // from current position
        {
            int spacesToMove;
            iss >> sub;
            std::istringstream(sub) >> spacesToMove;
            m_communityChest[i] = new CardMovePlayerBack(spacesToMove);
        }
    }


//Parse Each string in Chance vector and organise them into appropriate Cards
//Store each newly created Card into Chance Card Vector
  m_chance.resize(chanceCards.size());
  for(unsigned int i=0; i<chanceCards.size(); ++i)
  {
      line = chanceCards[i];
      std::istringstream iss(line);
      std::string sub;

      //First Sub-String of line is always flag
      iss >> sub;

      if (sub == "b") //Binary choice (e.g pay  1 0  or take a chance)
      {
          int moneyToRemove;
          iss >> sub;
          std::istringstream(sub) >> moneyToRemove;
          m_chance[i]= new CardChanceOrLoseMoney(moneyToRemove);
      }
```

```
        else if (sub == "f") //Get out of Jail Free
        {
          m_chance[i] = new CardGetOutOfJail();
        }

        else if (sub == "g") //Player RECIEVES money
        {
            int moneyToAdd;
            iss >> sub;
            std::istringstream(sub) >> moneyToAdd;
            m_chance[i] = new CardReceiveMoney(moneyToAdd);
        }

        else if (sub == "gp") //Receive money from EACH player
        {
            int moneyToAdd;
            iss >> sub;
            std::istringstream(sub) >> moneyToAdd;
            m_chance[i] = new CardReceiveMoneyFromPlayers(moneyToAdd);
        }

        else if (sub == "h") //Player has to perform repairs on property
        {
            int house, hotel;
            iss >> sub;
            std::istringstream(sub) >> house;
            iss >> sub;
            std::istringstream(sub) >> hotel;
            m_chance[i] = new CardStreetRepairs(house, hotel);
        }

        else if (sub == "j") //Go To Jail
        {
          m_chance[i] = new CardGoToJail();
        }

        else if (sub == "l") //Take money from player
        {
            int moneyToRemove;
            iss >> sub;
            std::istringstream(sub) >> moneyToRemove;
            m_chance[i] = new CardTakeMoney(moneyToRemove);
        }

        else if (sub == "m") //Move player TO SPECIFIED position
        {
            int boardPosition;
            iss >> sub;
            std::istringstream(sub) >> boardPosition;
            m_chance[i] = new CardMovePlayerToPosition(boardPosition);
        }

        else if (sub == "mb") //Move player back certain amount of spaces
                              // from current position
        {
            int spacesToMove;
            iss >> sub;
            std::istringstream(sub) >> spacesToMove;
            m_chance[i] = new CardMovePlayerBack(spacesToMove);
        }
     }
}


//-----------------------------------------------------------------------------
void CardsManager::print()const
{
    std::cout << "-------------------------------------------------\n";
    std::cout << "On a card." << std::endl;
}
```

```
//-----------------------------------------------------------------------------
CardsManager::~CardsManager()
{
    for(unsigned int i=0; i<m_communityChest.size();++i)
    {
        delete m_communityChest[i];
    }
    for(unsigned int i=0; i<m_chance.size();++i)
    {
        delete m_chance[i];
    }
}
```

Listing 21: CardStreetRepairs.hpp

```
#ifndef CARDSTREETREPAIRS_H
#define CARDSTREETREPAIRS_H

#include "Card.hpp"
#include <iostream>
#include <vector>

class CardStreetRepairs : public Card
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    CardStreetRepairs(
            const unsigned int i_houseRepair,
            const unsigned int i_hotelRepair
            );
    //-------------------------------------------------------------------------
    /// @brief the action that will be performed
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~CardStreetRepairs();

private:
    //-------------------------------------------------------------------------
    /// @brief price to repair a house
    //-------------------------------------------------------------------------
    unsigned int m_houseRepair;
    //-------------------------------------------------------------------------
    /// @brief price to repair a hotel
    //-------------------------------------------------------------------------
    unsigned int m_hotelRepair;

};

#endif // STREETREPAIRS_H
```

Listing 22: CardStreetRepairs.cpp

```
#include "CardStreetRepairs.hpp"

//-----------------------------------------------------------------------------
CardStreetRepairs::CardStreetRepairs(
        const unsigned int i_houseRepair,
        const unsigned int i_hotelRepair
        ):m_houseRepair(i_houseRepair),
          m_hotelRepair(i_hotelRepair)
{
}
```

```
//-----------------------------------------------------------------------------
void  CardStreetRepairs::action(
        PlayerManager &i_players
        )
{
  //NOTE:- Not complete implementation
    int numberOfHouses = 0;
    int numberOfHotels = 0;

    //Pseudo Code - [Calculate How Many Houses and Hotels There Are]

    i_players.takeBalance((numberOfHouses * m_houseRepair)
                        + (numberOfHotels * m_hotelRepair));
}


//-----------------------------------------------------------------------------
CardStreetRepairs::~CardStreetRepairs()
{}
```

Listing 23: CardTakeMoney.hpp

```
#ifndef CARDTAKEMONEY_H
#define CARDTAKEMONEY_H

#include "Card.hpp"

class CardTakeMoney :public Card
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    CardTakeMoney(unsigned int i_balance);
    //-------------------------------------------------------------------------
    /// @brief method that performs the action associated with that card
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~CardTakeMoney();

private:
    //-------------------------------------------------------------------------
    /// @brief the new position of the player
    //-------------------------------------------------------------------------
    int m_balanceToRemove;
};

#endif // CARDTAKEMONEY_H
```

Listing 24: CardTakeMoney.cpp

```
#include "CardTakeMoney.hpp"

//-----------------------------------------------------------------------------

CardTakeMoney::CardTakeMoney(
        unsigned int i_balance
        ):m_balanceToRemove(i_balance)
{
}

//-----------------------------------------------------------------------------
void CardTakeMoney::action(
        PlayerManager &i_players
        )
{
```

```
  //If Player can afford to pay rent, rent amount is removed from balance
    if(i_players.takeBalance(m_balanceToRemove))
    {
        std::cout << "Player " << i_players.getName() << " paid "
                  << m_balanceToRemove << "\n";
    }
    else
    {
        std::cout << "Player " << i_players.getName()
                  << "does not have enough money to pay\n";
    }
}

//------------------------------------------------------------------------
CardTakeMoney::~CardTakeMoney()
{
}
```

Listing 25: Dice.hpp

```
#ifndef DICE_H
#define DICE_H

#include <vector>

class Dice
{

public:
    //-------------------------------------------------------------------------------
    // constructor
    //-------------------------------------------------------------------------------
    Dice();
    //-------------------------------------------------------------------------------
    /// @brief method that returns the values of the dices
    //-------------------------------------------------------------------------------
    const std::vector<unsigned int> &getValues()const;
    //-------------------------------------------------------------------------------
    /// @brief method that returns the total sum of the values of the dices
    //-------------------------------------------------------------------------------
    unsigned int getTotal()const;
    //-------------------------------------------------------------------------------
    /// @brief method that prints the values of the dices
    //-------------------------------------------------------------------------------
    void print()const;
    //-------------------------------------------------------------------------------
    /// @brief returns true is the dices have the same number, otherwise false
    //-------------------------------------------------------------------------------
    bool isDouble()const;
    //-------------------------------------------------------------------------------
    /// @brief roll the dices
    //-------------------------------------------------------------------------------
    void roll();
    //-------------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------------
    ~Dice();

private:
    //-------------------------------------------------------------------------------
    /// @brief the values of the dices
    //-------------------------------------------------------------------------------
    std::vector<unsigned int> m_values;
};

#endif //DICE
```

Listing 26: Dice.cpp

```
#include <stdlib.h>
#include <iostream>
#include "Dice.hpp"


//--------------------------------------------------------------------------------
Dice::Dice()
{
    m_values.resize(2);
    roll();
}

//--------------------------------------------------------------------------------
void Dice::roll()
{
    m_values[0] = (rand() % 6) + 1;
    m_values[1] = (rand() % 6) + 1;
}

//--------------------------------------------------------------------------------
const std::vector<unsigned int> &Dice::getValues()const
{
    return m_values;
}

//--------------------------------------------------------------------------------
unsigned int Dice::getTotal()const
{
    // Testing: manually choose dice values
    /*
    unsigned int number = 1;
    std::cout << "Value: ";
    std::cin >> number;
    return number;
    */

    return (m_values[0] + m_values[1]);
}

//--------------------------------------------------------------------------------
void Dice::print()const
{
    std::cout << "Dice : " << m_values[0] << " " << m_values[1] << "\n";
}

//--------------------------------------------------------------------------------
bool Dice::isDouble() const
{
    return (m_values[0] == m_values[1]);
}


//--------------------------------------------------------------------------------
Dice::~Dice()
{}
```

Listing 27: Game.hpp

```
#ifndef GAME_H
#define GAME_H

class Player;
class Dice;

#include <iostream>
#include <vector>
#include "PlayerManager.hpp"
#include "Board.hpp"

class Game
```

```
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    Game();
    //-------------------------------------------------------------------------
    /// @brief method that plays the game
    //-------------------------------------------------------------------------
    void PlayGame();
    //-------------------------------------------------------------------------
    /// @brief method that resets a game
    //-------------------------------------------------------------------------
    void reset();
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~Game();

private:
    //-------------------------------------------------------------------------
    /// @brief method that takes a turn
    //-------------------------------------------------------------------------
    void takeTurn();

    //-------------------------------------------------------------------------
    /// @brief the board of the game
    //-------------------------------------------------------------------------
    Board m_board;
    //-------------------------------------------------------------------------
    /// @brief method that read the name of the next tile
    //-------------------------------------------------------------------------
    std::string readTilesName(
            const std::vector<std::string> &i_words,
            unsigned int *i_p
            );
    //-------------------------------------------------------------------------
    /// @brief all the Players of the Game
    //-------------------------------------------------------------------------
    PlayerManager m_players;
    //-------------------------------------------------------------------------
    /// @brief the game dice
    //-------------------------------------------------------------------------
    Dice m_dices;
};

#endif // GAME
```

Listing 28: Game.cpp

```
#include "Game.hpp"
#include "Dice.hpp"
#include <string>
#include <vector>
#include <climits>
#include <ctime>
#include <stdlib.h>


//-------------------------------------------------------------------------
Game::Game()
{}

//-------------------------------------------------------------------------
void Game::reset()
{

}

//-------------------------------------------------------------------------
```

```
void Game::PlayGame()
{
    std::cout << "\nNow we will start the game.\n" << std::endl;
    unsigned int numOfContinuousDoubles = 0;
    std::vector<unsigned int >numOfTimesRollDicesInJail;
    numOfTimesRollDicesInJail.resize(m_players.getNumOfPlayers());
    for(unsigned int i=0; i<numOfTimesRollDicesInJail.size();++i)
    {
        numOfTimesRollDicesInJail[i]=0;
    }
    while(m_players.howManyPlayersAreStillOntheGame()>1)
    {
        if(m_players.isCurrentPlayerInJail())
        {
            if(numOfTimesRollDicesInJail[m_players.getCurrentPlayer()]!=3)
            {
                std::cout << "\nPlayer "<< m_players.getName() << " is in Jail!\n";
                std::cout << "Do you want to pay 50 and go? (y/n)";
                std::string yesOrNo;
                std::cin >> yesOrNo;
                if(yesOrNo=="y")
                {
                    numOfTimesRollDicesInJail[m_players.getCurrentPlayer()]=0;
                    if(!m_players.takeBalance(50))
                    {
                        std::cout<< "You do not have enough money to pay!\n";
                        std::cout<< "Press 'R' to roll the dices!\n";
                        while (yesOrNo!= "R")
                        {
                            std::cin >> yesOrNo;
                        }
                        numOfTimesRollDicesInJail[m_players.getCurrentPlayer()]++;
                        m_dices.roll();
                        m_dices.print();
                        if(m_dices.isDouble())
                        {
                            m_players.movePositionBy(m_dices.getTotal());
                            m_board.action(m_players);
                        }
                    }
                    else
                    {
                        m_players.setJailed(false);
                        takeTurn();
                    }
                }else
                {
                    std::cout<< "Press 'R' to roll the dices!\n";
                    while (yesOrNo!= "R")
                    {
                        std::cin >> yesOrNo;
                    }
                    numOfTimesRollDicesInJail[m_players.getCurrentPlayer()]++;
                    m_dices.roll();
                    m_dices.print();
                    if(m_dices.isDouble())
                    {
                        m_players.movePositionBy(m_dices.getTotal());
                        m_board.action(m_players);
                    }
                }
                m_players.moveToNextPlayer();
            }
            else
            {
                numOfTimesRollDicesInJail[m_players.getCurrentPlayer()]=0;
                std::cout << "You have to pay 50 and get out of Jail!\n";
                if(!m_players.takeBalance(50))
                {
                    std::cout << "You do not have enough money to pay!\n";
                    m_players.withdrawGame();
```

```
                m_players.moveToNextPlayer();
            }
            else
            {
                m_players.setJailed(false);
                std::cout<< "Press 'R' to roll the dices!\n";
                std::string answer="";
                while (answer!= "R")
                {
                    std::cin >> answer;
                }
                takeTurn();
                m_players.moveToNextPlayer();
            }
        }
    }else
    {
        std::cin.clear();
        std::cin.ignore(INT_MAX, '\n');
        std::string answer;
        std::cout << "\n   ***  Player: " << m_players.getName()
                  << ",  Balance = " << m_players.getBalance() <<"  ***\n";

       std::cout  << "Press B to build houses, R to roll and play, or "
                  << " W to Withdraw the game \n";
       while(answer!="R"&& answer!="B" && answer!="W")
       {
           std::cin >> answer;
       }

        if(answer == "R")
        {
            takeTurn();
        }
        else if(answer == "W")
        {
            m_players.withdrawGame();
        }
        else  // buy houses
        {
            m_board.buildHouses(m_players);
            std::cout << "Press R to roll and play\n";
            while(answer!="R")
            {
                std::cin >> answer;
            }
            takeTurn();
        }
        if(m_dices.isDouble())
        {
            if(numOfContinuousDoubles==3)
            {
                m_players.setJailed(true);
                numOfContinuousDoubles = 0;
            }
            numOfContinuousDoubles++;
            m_board.action(m_players);
        }
        else
        {
            m_board.action(m_players);
            m_players.moveToNextPlayer();
        }
    }
}
    m_players.printWinner();
}

//--------------------------------------------------------------------------
void Game::takeTurn()
{
```

```
    m_dices.roll();
    m_dices.print();
    m_players.movePositionBy(m_dices.getTotal());
    std::cout << "You have reach the following Tile:\n";
    m_board.print(m_players.getCurrentPlayersPosition());
}

//--------------------------------------------------------------------------
Game::~Game()
{
}
```

Listing 29: GetACardTile.hpp

```
#ifndef GETACARDTILE_H
#define GETACARDTILE_H

#include "CardsManager.hpp"
#include "Tile.hpp"

class GetACardTIle  : public Tile
{
public:
    //--------------------------------------------------------------------------
    /// @brief default constructor
    //--------------------------------------------------------------------------
    GetACardTIle(
            const std::string &m_name,
            CardsManager & i_cardDesks
            );
    //--------------------------------------------------------------------------
    /// @brief method that does the action =p
    /// @param[in] i_players the players of the game
    //--------------------------------------------------------------------------
    void action(PlayerManager &i_players);
    //--------------------------------------------------------------------------
    /// @brief method that resets all its values to the default ones
    //--------------------------------------------------------------------------
    void reset();

private:
    //--------------------------------------------------------------------------
    /// @brief the card decks
    //--------------------------------------------------------------------------
    CardsManager &m_cardDecks;
};

#endif // GETACARDTILE_H
```

Listing 30: GetACardTile.cpp

```
#include "GetACardTIle.hpp"

//--------------------------------------------------------------------------
GetACardTIle::GetACardTIle(
        const std::string &m_name,
        CardsManager &i_cardDesks
        ):Tile(m_name),
          m_cardDecks(i_cardDesks)
{
}

//--------------------------------------------------------------------------
void GetACardTIle::action(PlayerManager &i_players)
{
    m_cardDecks.action(i_players,m_name);
}

//--------------------------------------------------------------------------
```

```
void GetACardTIle::reset()
{
}
```

Listing 31: GroupOfProperties.hpp

```
#ifndef GROUPOFPROPERTIES_H
#define GROUPOFPROPERTIES_H

#include <iostream>
#include "Tile.hpp"
#include "PlayerManager.hpp"

class GroupOfProperties
{
    friend class GroupsManager;
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    /// @param[in] i_colour the colour of the property
    //-------------------------------------------------------------------------
    GroupOfProperties(const std::string &i_colour);
    //-------------------------------------------------------------------------
    /// @brief constructor
    /// @param[in] i_colour the colour of the property
    /// @param[in] i_tile first tile to be added to the group
    //-------------------------------------------------------------------------
    GroupOfProperties(const std::string &i_colour, Tile *i_tile);
    //-------------------------------------------------------------------------
    /// @brief copy constructor
    /// @param[in] i_group group to be copied
    //-------------------------------------------------------------------------
    GroupOfProperties(const GroupOfProperties& i_group);
    //-------------------------------------------------------------------------
    /// @brief method that adds a tile to the end of m_tiles
    ///        returns true if item has been added
    ///        returns false if item does not belong to that group
    /// @param[in] i_tile pointer to the tile added to the group
    /// @param[in] i_colour the colour of the tile
    //-------------------------------------------------------------------------
    bool addTile(const std::string i_colour, Tile *i_tile);
    //-------------------------------------------------------------------------
    /// @brief method that returns house price
    //-------------------------------------------------------------------------
    unsigned int getHousePrice()const;
    //-------------------------------------------------------------------------
    /// @brief returns the number of properties the given Player owns
    /// @param[in] the index of the player of our interest
    //-------------------------------------------------------------------------
    unsigned int getNumOfOwns(unsigned int i_player)const;
    //-------------------------------------------------------------------------
    /// @brief method that prints all the properties that belong to that group
    //-------------------------------------------------------------------------
    void print()const;
    //-------------------------------------------------------------------------
    /// @brief method that build a given number of houses
    /// @brief returns number of houses that have been successfully build
    /// @param[in] i_num: number of houses to be build
    //-------------------------------------------------------------------------
    unsigned int buildHouses(unsigned int i_num,PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief method that remove houses from properties
    ///        returns the amount of houses that has been sucessfully sold
    /// @param[in] i_number number of houses to be sold
    //-------------------------------------------------------------------------
    unsigned int sellHouses(unsigned int i_number);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~GroupOfProperties();
```

```
private:
    //--------------------------------------------------------------------------
    /// @brief the colour of the property, for utilities is equal to "utility"
    ///        and for stations is equalt to "station"
    //--------------------------------------------------------------------------
    std::string m_colour;
    //--------------------------------------------------------------------------
    /// @brief pointers to the tiles that relates with each other
    //--------------------------------------------------------------------------
    std::vector<Tile *> m_tiles;
    //--------------------------------------------------------------------------
    /// @brief indicates where the next house will be build
    //--------------------------------------------------------------------------
    unsigned int m_whereToBuild;
};

#endif // PROPERTIESRELATION_H
```

Listing 32: GroupOfProperties.cpp

```
#include "GroupOfProperties.hpp"

//------------------------------------------------------------------------------
GroupOfProperties::GroupOfProperties(
        const std::string &i_colour
        ):m_colour(i_colour),
          m_whereToBuild(0)
{
}

//------------------------------------------------------------------------------
GroupOfProperties::GroupOfProperties(
        const std::string &i_colour,
        Tile *i_tile
        ):m_colour(i_colour),
          m_whereToBuild(0)
{
    m_tiles.push_back(i_tile);
}

//------------------------------------------------------------------------------
GroupOfProperties::GroupOfProperties(const GroupOfProperties &i_group)
{
    m_colour=i_group.m_colour;
    for(unsigned int i=0; i<i_group.m_tiles.size();++i)
    {
        m_tiles.push_back(i_group.m_tiles[i]);
    }

}

//------------------------------------------------------------------------------
void GroupOfProperties::print()const
{
    std::cout<<"The following tiles belong to group: " << m_colour << "\n";
    for(unsigned int i=0; i<m_tiles.size(); ++i)
    {
        m_tiles[i]->print();
    }
}

//------------------------------------------------------------------------------
bool GroupOfProperties::addTile(const std::string i_colour, Tile *i_tile)
{
    if(i_colour==m_colour)
    {
        m_tiles.push_back(i_tile);
        return true;
    }
```

```
    else
    {
        return false;
    }
}

//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
unsigned int GroupOfProperties::getHousePrice()const
{
    return m_tiles[0]->getHousePrice();
}

//---------------------------------------------------------------------------
unsigned int GroupOfProperties::getNumOfOwns(unsigned int i_player)const
{
    unsigned int num =0;
    for(unsigned int i=0; i<m_tiles.size(); ++i)
    {
        if(m_tiles[i]->getOwner()==i_player)
        {
            num++;
        }
    }
    return num;
}

//---------------------------------------------------------------------------
unsigned int GroupOfProperties::buildHouses(
        unsigned int i_num,
        PlayerManager &i_players
        )
{
    unsigned int numOfHousesSuccessfullyBuild = 0;
    const unsigned int costOfAHouse = 10;//m_tiles[0]->getHousePrice();
    if(m_colour=="station" || m_colour=="utility")
    {
        return 0; // you cannot build houses on utilities and stations
    }
    for(unsigned int i=0; i<i_num ; ++i)
    {
        if(i_players.takeBalance(costOfAHouse))
        {
            if(m_tiles[m_whereToBuild]->buildHouse())
            {
                // there is a hotel
                i_players.addBalance(costOfAHouse);
                numOfHousesSuccessfullyBuild++;
            }
            m_whereToBuild = (m_whereToBuild+1)%m_tiles.size();
        }
        else
        {
            // player does not have enough money to buy the rest of the houses
            return numOfHousesSuccessfullyBuild;
        }
    }
    return numOfHousesSuccessfullyBuild;
}


//---------------------------------------------------------------------------
GroupOfProperties::~GroupOfProperties()
{}
```

Listing 33: GroupsManager.hpp

```
#ifndef GROUPSMANAGER_H
```

```
#define GROUPSMANAGER_H

#include <iostream>
#include <vector>
#include "GroupOfProperties.hpp"

class GroupsManager
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    GroupsManager();
    //-------------------------------------------------------------------------
    /// @brief method that adds a tile to the relations class
    /// @param[in] i_colour the colour of the tile
    /// @param[in] i_position the positiong of the tile on the board;
    //-------------------------------------------------------------------------
    void addTile(const std::string &i_colour,Tile *i_tile);
    //-------------------------------------------------------------------------
    /// @brief method that builds houses to a group of properties
    //-------------------------------------------------------------------------
    unsigned int buildHouses(
            PlayerManager &i_players
            );
    //-------------------------------------------------------------------------
    /// @brief returns how many stations the given owner has
    /// @param[in] i_owner the owner that owns stations
    //-------------------------------------------------------------------------
    unsigned int getNumOfStations(unsigned int i_owner)const;
    //-------------------------------------------------------------------------
    /// @brief method that prints all the tiles in groups
    //-------------------------------------------------------------------------
    void print()const;
    //-------------------------------------------------------------------------
    /// @brief method that sells houses from a group of properties
    ///        returns amount of money that the player gets
    //-------------------------------------------------------------------------
    unsigned int sellHouses(std::string i_colour, unsigned int i_number);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~GroupsManager();

private:
    //-------------------------------------------------------------------------
    /// @brief method that checks whether the given owner has permission to
    ///        build or sell houses to that group of properties
    /// @brief returns the index of the group or -1 if the owner is not allowed
    /// @param[in] i_owner the owner that wants to build houses
    /// @param[in] i_colour the colour of the group that the owner wants to
    ///                     build houses on or to sell houses
    //-------------------------------------------------------------------------
    unsigned int allowedToBuildOrSellHouses(
            unsigned int i_owner,
            const std::string &i_colour
            );
    //-------------------------------------------------------------------------
    /// @brief all the groups of properties ie, stations and blue properties
    //-------------------------------------------------------------------------
    std::vector<GroupOfProperties> m_groups;

};

#endif // PROPERTIESRELATIONS_H
```

Listing 34: GroupsManager.cpp

```
#include "GroupsManager.hpp"
#include <stdlib.h>
```

```
//--------------------------------------------------------------------------------
GroupsManager::GroupsManager()
{
}

//--------------------------------------------------------------------------------
void GroupsManager::addTile(
        const std::string &i_colour,
        Tile *i_tile
        )
{
    bool isTileAdded = false; //indicated whether the tile is addes to a group
    for (unsigned int i=0; i<m_groups.size(); ++i)
    {
        isTileAdded+=m_groups[i].addTile(i_colour,i_tile);
    }
    if(isTileAdded==false)
    {
        m_groups.push_back(GroupOfProperties(i_colour,i_tile));
    }
}

//--------------------------------------------------------------------------------
void GroupsManager::print()const
{
    for(unsigned int i=0; i<m_groups.size(); ++i)
    {
        std::cout << "**********************************\n";
        m_groups[i].print();
    }
}

//--------------------------------------------------------------------------------
unsigned int GroupsManager::buildHouses(
         PlayerManager &i_players
        )
{
    std::vector<int> groups; // groups player is allowed to build houses
    for(unsigned int i=0; i<m_groups.size();++i)
    {
        if(m_groups[i].m_colour!="utility"&& m_groups[i].m_colour!="station" &&
           m_groups[i].m_tiles.size() ==
                m_groups[i].getNumOfOwns(i_players.getCurrentPlayer()))
        {
            groups.push_back(i);
        }
    }
    if(groups.size()==0)
    {
        std::cout << "You are not allowed to buy houses yet\n";
        return 0;
    }
    else
    {
        std::cout << "Please choose where to build houses by pressing"
                  << " the corresponding number:\n";
        for(unsigned int i=0; i<groups.size();++i)
        {
            std::cout<<i<<": build on "<< m_groups[groups[i]].m_colour<<" for "
                     <<m_groups[groups[i]].getHousePrice() << " each\n";
        }
        std::cout<< "Or type 'Q' if you have changed your mind\n";
        std::string answer;
        std::cin >> answer;
        if(answer!="Q")
        {
            const unsigned int whereTobuild = atoi(answer.c_str());
            if(whereTobuild>groups.size())
            {
```

```
                std::cout << "This number was not given in the choice list\n";
                return 0;
            }
            std::cout << "Please give number of houses to be build:\n";
            std::cin >> answer;
            const unsigned int numOfHousesToBuild = atoi(answer.c_str());
            std::cout << "ITS IS EXPECTED TO BUILD " << numOfHousesToBuild << "\n";
            unsigned int numOfHousesBuild = m_groups[groups[whereTobuild]]
                    .buildHouses(numOfHousesToBuild,i_players);
            std::cout << numOfHousesBuild << " has been successfully build\n";
        }
        else
        {
            // Player has changed his mind
        }
    }
   return 0;
}

//--------------------------------------------------------------------------------
unsigned int GroupsManager::getNumOfStations(unsigned int i_owner) const
{
    unsigned int num = 0;
    for(unsigned int i=0; i<m_groups.size(); ++i)
    {
        if(m_groups[i].m_colour == "station")
        {
            for(unsigned int j=0; j<m_groups[i].m_tiles.size();++j)
            {
                if(m_groups[i].m_tiles[j]->getOwner()==i_owner)
                {
                    num++;
                }
            }
            break;
        }
    }
    return num;
}

//--------------------------------------------------------------------------------
unsigned int GroupsManager::sellHouses(
        std::string /*i_colour*/,
        unsigned int /*i_number*/
        )
{
    return 1;
}


//--------------------------------------------------------------------------------
GroupsManager::~GroupsManager()
{}
```

Listing 35: NormalProperty.hpp

```
#ifndef NORMALPROPERTY_H
#define NORMALPROPERTY_H

#include "Property.hpp"
#include <vector>

class NormalProperty: public Property
{
public:
    //--------------------------------------------------------------------------
    /// @brief default constructor
    //--------------------------------------------------------------------------
    NormalProperty(const std::string &i_name,
                   unsigned int i_price,
```

```
                    unsigned int i_housePrice,
                    const std::vector<unsigned int> &i_rentPrices
                    );
    //-------------------------------------------------------------------------
    /// @brief method that prints all the information about the properties
    //-------------------------------------------------------------------------
    void printExtras()const;
    //-------------------------------------------------------------------------
    /// @brief method that returns the owner of the property
    //-------------------------------------------------------------------------
    unsigned int getOwner()const;
    //-------------------------------------------------------------------------
    /// @brief method that returns how much it cost to build a house
    //-------------------------------------------------------------------------
    unsigned int getHousePrice()const;
    //-------------------------------------------------------------------------
    /// @brief method that resets all its values to the default ones
    //-------------------------------------------------------------------------
    void resetExtras();
    //-------------------------------------------------------------------------
    /// @brief method that builds a house to a property
    //-------------------------------------------------------------------------
    bool buildHouse();
    //-------------------------------------------------------------------------
    /// @brief returns the number of houses a property has
    //-------------------------------------------------------------------------
    unsigned int getNumOfHouses()const;
    //-------------------------------------------------------------------------
    /// @brief method that does the action =p
    /// @param[in] i_player the player that have reached that specific tile
    //-------------------------------------------------------------------------
    void payRent(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~NormalProperty();

private:
    //-------------------------------------------------------------------------
    /// @brief how much does a house cost
    //-------------------------------------------------------------------------
    double m_housePrice;
    //-------------------------------------------------------------------------
    /// @brief the number of houses the propertry has (0-5)
    //-------------------------------------------------------------------------
    unsigned short int m_numOfHouses;
    //-------------------------------------------------------------------------
    /// @brief the prices of renting a the property depending
    //-------------------------------------------------------------------------
    std::vector<unsigned int> m_rentPrices;


};

#endif // NORMALPROPERTY_H
```

Listing 36: NormalProperty.cpp

```
#include "NormalProperty.hpp"

//-----------------------------------------------------------------------
NormalProperty::NormalProperty(
        const std::string &i_name,
        unsigned int i_price,
        unsigned int i_housePrice,
        const std::vector<unsigned int> &i_rentPrices
        ):Property(i_name,i_price),
          m_housePrice(i_housePrice),
          m_numOfHouses(0)
{
```

```
    m_rentPrices.resize(6);
    for(unsigned int i=0; i<6; ++i)
    {
        m_rentPrices[i] = i_rentPrices[i];
    }
}

//------------------------------------------------------------------------
void NormalProperty::printExtras()const
{
    std::cout << "House Price: " << m_housePrice << "\n";
    std::cout << "Rent Prices: ";
    for(unsigned int i=0; i<6; ++i)
    {
        std::cout << m_rentPrices[i] << " ";
    }
    std::cout << "\n-------------------------------------------------\n";
    std::cout << "\n";
}

//------------------------------------------------------------------------
unsigned int NormalProperty::getOwner()const
{
    return m_owner;
}

//------------------------------------------------------------------------
unsigned int NormalProperty::getHousePrice()const
{
    return m_housePrice;
}

//------------------------------------------------------------------------
unsigned int NormalProperty::getNumOfHouses()const
{
    return m_numOfHouses;
}

//------------------------------------------------------------------------
bool NormalProperty::buildHouse()
{
    if(m_numOfHouses<5)
    {
        m_numOfHouses++;
        return true;
    }
    else
    {
        return false; // a hotel exists
    }
}

//------------------------------------------------------------------------
void NormalProperty::payRent(
        PlayerManager &i_players
        )
{
    if(i_players.takeBalance(m_rentPrices[m_numOfHouses]))
    {
        i_players.addBalance(m_rentPrices[m_numOfHouses],m_owner);
        std::cout << "Rent paid: " << m_rentPrices[m_numOfHouses] << std::endl;
    }
    else
    {
        std::cout << "Player does not have enough money to pay\n";
    }
}


//------------------------------------------------------------------------
void NormalProperty::resetExtras()
```

```
{
    m_numOfHouses = 0;
}

//-------------------------------------------------------------------------
NormalProperty::~NormalProperty()
{}
```

Listing 37: Order.hpp

```
#ifndef ORDER_H
#define ORDER_H

#include "Tile.hpp"

class Order: public Tile
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    /// @param[in] i_name: the name of the tile
    //-------------------------------------------------------------------------
    Order(const std::string &i_name, const std::string &flag);
    //-------------------------------------------------------------------------
    /// @brief default constructor
    /// @param[in] i_name: the name of the tile
    /// @param[in] i_money: money that needs to be paid
    //-------------------------------------------------------------------------
    Order(
            const std::string &i_name,
            const std::string &flag,
            unsigned int i_money
            );
    //-------------------------------------------------------------------------
    /// @brief method that prints all the information about the Order Tile
    //-------------------------------------------------------------------------
    void print()const;
    //-------------------------------------------------------------------------
    /// @brief method that resets all its values to the default ones
    //-------------------------------------------------------------------------
    void reset();
    //-------------------------------------------------------------------------
    /// @brief method that does the action =p
    /// @param[in] i_player the player that have reached that specific tile
    //-------------------------------------------------------------------------
    void action(PlayerManager &i_players );
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~Order();

private:
    //-------------------------------------------------------------------------
    /// @brief indicates action to be perform
    //-------------------------------------------------------------------------
    std::string m_flag;
    //-------------------------------------------------------------------------
    /// @brief amount of money in case money has to be paid
    //-------------------------------------------------------------------------
    unsigned int m_money;
};

#endif // ORDER_H
```

Listing 38: Order.cpp

```
#include "Order.hpp"

//-------------------------------------------------------------------------
```

```
Order::Order(
        const std::string &i_name,
        const std::string &flag
        ):Tile(i_name),
          m_flag(flag),
          m_money(0)
{
}

//--------------------------------------------------------------------------------
Order::Order(
            const std::string &i_name,
            const std::string &flag,
            unsigned int i_money
            ):Tile(i_name),
              m_flag(flag),
              m_money(i_money)
{
}

//--------------------------------------------------------------------------------
void Order::print()const
{
    std::cout << "--------------------------------------------------\n";
    std::cout << "Tile's name: " << m_name << "\n";
    if( m_flag == "p")
    {
        std::cout << "Amount to be paid: " << m_money << "\n";
    }
    std::cout << "--------------------------------------------------\n\n";
}

//--------------------------------------------------------------------------------
void Order::action(
        PlayerManager & i_players
        )
{
    if(m_flag=="p") // pay money
    {
        std::cout << i_players.getName() << " has to pay " << m_money << "\n";
        i_players.takeBalance(m_money);
    }
    else if(m_flag=="j") // go to jail
    {
        std::cout << "You are going to JAIL!\n";
        i_players.setJailed(true);
    }
    else if (m_flag == "f") // free parking, go, or visiting jail
    {
    }
    else
    {
        //wrong flag
    }
}

//--------------------------------------------------------------------------
void Order::reset()
{
}

//--------------------------------------------------------------------------
Order::~Order()
{}
```

Listing 39: Player.hpp

```
#ifndef PLAYER
#define PLAYER
```

```
#include <string>
#include <vector>

class Player
{
    friend class PlayerManager;
public:
    //-------------------------------------------------------------------------
    /// default constructor
    //-------------------------------------------------------------------------
    Player();
    //-------------------------------------------------------------------------
    /// constructor
    //-------------------------------------------------------------------------
    Player(const std::string &name);
    //-------------------------------------------------------------------------
    /// @brief method that sets the name of a Player
    //-------------------------------------------------------------------------
    void setName(const std::string &i_name);
    //-------------------------------------------------------------------------
    /// @brief takes an amount of money from the balance of the Player
    //-------------------------------------------------------------------------
    bool takeBalance(unsigned int i_amount);
    //-------------------------------------------------------------------------
    /// @brief method that returns how much the possessions of the player worth
    //-------------------------------------------------------------------------
    unsigned int getPossessionsValue() const;
    //-------------------------------------------------------------------------
    /// @brief method that resets all the members of the Player
    //-------------------------------------------------------------------------
    void reset();
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~Player();


private:
    //-------------------------------------------------------------------------
    /// @brief the name of the Player
    //-------------------------------------------------------------------------
    std::string m_name;
    //-------------------------------------------------------------------------
    /// @brief the balance of the Player
    //-------------------------------------------------------------------------
    unsigned int m_balance;
    //-------------------------------------------------------------------------
    /// @brief the position of the Player on the Board
    //-------------------------------------------------------------------------
    unsigned int m_position;
    //-------------------------------------------------------------------------
    /// @brief whether or not the player is in jail, -1 if not in jail else
    ///        represents how many time you can still try to get out free
    //-------------------------------------------------------------------------
    int m_isJailed;
    //-------------------------------------------------------------------------
    /// @brief number of "get out of Jail" Cards the player owns
    //-------------------------------------------------------------------------
    unsigned int m_numOfGetOutOfJailCards;

};

#endif //PLAYER
```

Listing 40: Player.cpp

```
#include <string>
#include <vector>
#include "Player.hpp"
#include <iostream>
```

```
//--------------------------------------------------------------------------------
Player::Player(
        ):m_balance(1500),
          m_position(0),
          m_isJailed(0),
    m_numOfGetOutOfJailCards(0)
{}

//--------------------------------------------------------------------------------
Player::Player(
            const std::string &name
            ): m_name(name),
               m_balance(1500),
               m_position(0),
               m_isJailed(false)
{}

bool Player::takeBalance(unsigned int i_amount)
{
    {
        if(m_balance<i_amount)
        {
            return false;
        }
        else
        {
            m_balance -= i_amount;
            return true;
        }
    }
}

//--------------------------------------------------------------------------------
void Player::setName(const std::string &i_name)
{
    m_name=i_name;
}

//--------------------------------------------------------------------------------
void Player::reset()
{
    m_balance = 1500;
    m_position = 0;
    m_isJailed = false;
}

//--------------------------------------------------------------------------------
unsigned int Player::getPossessionsValue()const
{
    return m_balance;
}

//--------------------------------------------------------------------------------
Player::~Player()
{}
```

Listing 41: PlayerManager.hpp

```
#ifndef PLAYERS_H
#define PLAYERS_H

#include "Player.hpp"
#include "Dice.hpp"

class PlayerManager
{
public:
    //--------------------------------------------------------------------------------
    /// @brief default constructor
```

```
//-----------------------------------------------------------------------------
PlayerManager();
//-----------------------------------------------------------------------------
/// @brief constructor
/// @param[in] i_numOfPlayers: how many players are playing
//-----------------------------------------------------------------------------
PlayerManager(unsigned int i_numOfPlayers);
//-----------------------------------------------------------------------------
/// @brief method that returns the position on the board of
///        the current player
//-----------------------------------------------------------------------------
unsigned int getCurrentPlayersPosition()const;
//-----------------------------------------------------------------------------
/// @brief method that returns number of players that are still on the game
//-----------------------------------------------------------------------------
unsigned int howManyPlayersAreStillOntheGame()const;
//-----------------------------------------------------------------------------
/// @brief set the number of Players and there game
/// @param[in] i_numOfPlayers: how many players are playing
//-----------------------------------------------------------------------------
void setPlayers(unsigned int i_numOfPlayers);
//-----------------------------------------------------------------------------
/// @brief method that prints all players that are still on the game
///        starting with the person that owns mos
//-----------------------------------------------------------------------------
void printWinner()const;
//-----------------------------------------------------------------------------
/// @brief reset Players
//-----------------------------------------------------------------------------
void resetAllPlayers();
//-----------------------------------------------------------------------------
/// @brief method that changes the current player to the next player
//-----------------------------------------------------------------------------
void moveToNextPlayer();
//-----------------------------------------------------------------------------
/// @brief method that returns the Balance of the Player
//-----------------------------------------------------------------------------
unsigned int getBalance();
//-----------------------------------------------------------------------------
/// @brief takes an amount of money from the balance of the Player
//-----------------------------------------------------------------------------
bool takeBalance(unsigned int i_amount);
//-----------------------------------------------------------------------------
/// @brief method that adds an amount of money to the balance of the Player
//-----------------------------------------------------------------------------
void addBalance(unsigned int i_amount);
//-----------------------------------------------------------------------------
/// @brief method that the current player quits the game
/// @brief it returns how much he owns in case he have to pay someone
//-----------------------------------------------------------------------------
unsigned int withdrawGame();
//-----------------------------------------------------------------------------
/// @brief method that adds an amount of money to a specific Player
/// @param[in] i_amount: amount to be added
/// @param[in] i_player: the player that gets the money
//-----------------------------------------------------------------------------
void addBalance(unsigned int i_amount, unsigned int i_player);
//-----------------------------------------------------------------------------
/// @brief method that returns the current position of the current Player
//-----------------------------------------------------------------------------
unsigned int getPosition() const;
//-----------------------------------------------------------------------------
/// @brief method that moves the current player to a new position
/// @param[in] i_amount how many tiles the current Player will be moved
//-----------------------------------------------------------------------------
void movePositionBy(unsigned int i_amount);
//-----------------------------------------------------------------------------
/// @brief method that sets the positions of the current player
/// @param[in] i_position the new Position of the current Player
//-----------------------------------------------------------------------------
void setPosition(unsigned int i_position);
```

```
    //-------------------------------------------------------------------------
    /// @brief method that sends the current player to jail
    //-------------------------------------------------------------------------
    void setJailed(bool jailed);
    //-------------------------------------------------------------------------
    /// @brief method that returns the current player
    //-------------------------------------------------------------------------
    unsigned int getCurrentPlayer()const;
    //-------------------------------------------------------------------------
    /// @brief method that returns the name of the current Player
    //-------------------------------------------------------------------------
    const std::string &getName()const;
    //-------------------------------------------------------------------------
    /// @brief method that gets an amount of money from each player
    /// @param[in] i_amount amount to be removed from players
    //-------------------------------------------------------------------------
    unsigned int getMoneyFromEachPlayer(unsigned int i_amount);
    //-------------------------------------------------------------------------
    /// @brief method tha returns true if current Player is in Jail
    //-------------------------------------------------------------------------
    bool isCurrentPlayerInJail()const;
    //-------------------------------------------------------------------------
    /// @brief method that returns the number of Players
    //-------------------------------------------------------------------------
    unsigned int getNumOfPlayers()const;
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~PlayerManager();

private:
    //-------------------------------------------------------------------------
    /// @brief all the Players of the game
    //-------------------------------------------------------------------------
    std::vector<Player> m_players;
    //-------------------------------------------------------------------------
    /// @brief the number of the current Player
    //-------------------------------------------------------------------------
    unsigned int m_currentPlayer;
    //-------------------------------------------------------------------------
    /// @brief num of players that have lost the game
    //-------------------------------------------------------------------------
    unsigned int m_numOfLoosers;
    //-------------------------------------------------------------------------
    /// @brief the dices
    //-------------------------------------------------------------------------
    Dice m_dices;
};

#endif // PLAYERS_H
```

Listing 42: PlayerManager.cpp

```
#include "PlayerManager.hpp"
#include <iostream>
#include <string>
#include <vector>
#include <climits>
#include <ctime>
#include <stdlib.h>

//-----------------------------------------------------------------------------
PlayerManager::PlayerManager(
        ):m_currentPlayer(0),
          m_numOfLoosers(0)
{
    srand((unsigned)time(0));
    unsigned int numPlayers = 2;
    std::cout << "Enter number of players (2 - 6): ";
    std::cin >> numPlayers;
```

```
    while (numPlayers < 2 || numPlayers > 6) {
        std::cin.clear();
        std::cin.ignore(INT_MAX, '\n');
        std::cout << "Please enter a number between 2 and 6: ";
        std::cin >> numPlayers;
    }
    m_players.resize(numPlayers);
    setPlayers(numPlayers);
}

//--------------------------------------------------------------------------------
PlayerManager::PlayerManager(
        unsigned int i_numOfPlayers
        ):m_currentPlayer(0),
          m_numOfLoosers(0)
{
    setPlayers(i_numOfPlayers);
}

//--------------------------------------------------------------------------------
unsigned int PlayerManager::getNumOfPlayers()const
{
    return m_players.size();
}

//--------------------------------------------------------------------------------
void PlayerManager::setPlayers(unsigned int i_numOfPlayers)
{
    m_players.resize(i_numOfPlayers);
    std::string currentName;
    for (unsigned int i=0;i<i_numOfPlayers;i++) {
        std::cout << "What's your name, player " << i+1 << "? ";
        std::cin >> currentName;
        m_players[i].setName(currentName);
    }
    std::cout << std::endl;

    std::cout << "Now we'll roll the dice to see who goes first"
              << std::endl;

    int oldwinner = 0;
    int winner = 0;
    int windex = -1;
    std::cin.clear();
    std::cin.ignore(INT_MAX, '\n');

    for (unsigned int i=0;i<m_players.size();i++) {
        std::cout << "Press ENTER to roll, " << m_players[i].m_name
                  << std::endl;
        std::cin.get();
        m_dices.roll();
        winner = m_dices.getTotal();
        std::cout << m_players[i].m_name << " rolled: " << winner
                  << std::endl <<std::endl;
        if (winner > oldwinner) windex = i;
        oldwinner = winner;
    }

    std::cout << m_players[windex].m_name << " goes first." << std::endl;

    m_currentPlayer = windex;
    std::cout << std::endl;
}

//--------------------------------------------------------------------------------
unsigned int PlayerManager::withdrawGame()
{
    const unsigned int balance = m_players[m_currentPlayer].m_balance;
    m_players[m_currentPlayer].m_balance = 0;
    return balance;
}
```

```
//-----------------------------------------------------------------------------
void PlayerManager::printWinner()const
{
    std::string winner=m_players[0].m_name;
    unsigned int maxPossesionsValue = 0;
    for(unsigned int i=0; i<m_players.size(); ++i)
    {
        if(m_players[i].m_balance>0)
        {
            const unsigned int currentPossesionValues =
                m_players[i].getPossessionsValue();
            if(maxPossesionsValue<currentPossesionValues)
            {
                maxPossesionsValue = currentPossesionValues;
                winner = m_players[i].m_name;
            }
        }
    }
    std::cout << "The winner is " << winner << " and his balance is "
              << maxPossesionsValue << "\n";
}

//-----------------------------------------------------------------------------
void PlayerManager::resetAllPlayers()
{
    for(unsigned int i=0; i<m_players.size(); ++i)
    {
        m_players[i].reset();
    }
}

//-----------------------------------------------------------------------------
unsigned int PlayerManager::getCurrentPlayersPosition()const
{
   return m_players[m_currentPlayer].m_position;
}

//-----------------------------------------------------------------------------
unsigned int PlayerManager::howManyPlayersAreStillOntheGame()const
{
    unsigned int num = 0;
    for(unsigned int i=0; i<m_players.size(); ++i)
    {
        if(m_players[i].m_balance>0)
        {
            num+=1;
        }
    }
    return num;
}

//-----------------------------------------------------------------------------
bool PlayerManager::isCurrentPlayerInJail()const
{
    return m_players[m_currentPlayer].m_isJailed;
}

//-----------------------------------------------------------------------------
void PlayerManager::moveToNextPlayer()
{
    m_currentPlayer = (m_currentPlayer+1)%m_players.size();
    int numOfPlayers = m_players.size();
    while(m_players[m_currentPlayer].m_balance==0)
    {
        m_currentPlayer = (m_currentPlayer+1)%m_players.size();
        numOfPlayers--;
        if(numOfPlayers==0)
        {
            break;
        }
```

```
    }
}

//--------------------------------------------------------------------------------
const std::string &PlayerManager::getName() const
{
    return m_players[m_currentPlayer].m_name;
}

//--------------------------------------------------------------------------------
unsigned int PlayerManager::getBalance()
{
    return m_players[m_currentPlayer].m_balance;
}

//--------------------------------------------------------------------------------
bool PlayerManager::takeBalance(unsigned int i_amount)
{
    if(m_players[m_currentPlayer].m_balance<i_amount )
    {
        return false;
    }
    else
    {
        m_players[m_currentPlayer].m_balance -= i_amount;
        return true;
    }
}

//--------------------------------------------------------------------------------
void PlayerManager::addBalance(unsigned int i_amount)
{
    m_players[m_currentPlayer].m_balance+=i_amount;
}

//--------------------------------------------------------------------------------
void PlayerManager::addBalance(unsigned int i_amount, unsigned int i_player)
{
    m_players[i_player].m_balance+=i_amount;
}

//--------------------------------------------------------------------------------
unsigned int PlayerManager::getPosition() const
{
    return m_players[m_currentPlayer].m_position;
}

//--------------------------------------------------------------------------------
void PlayerManager::movePositionBy(unsigned int i_amount)
{
    unsigned int oldPosition = m_players[m_currentPlayer].m_position;

    m_players[m_currentPlayer].m_position =
            (m_players[m_currentPlayer].m_position + i_amount) % 40;
    if (m_players[m_currentPlayer].m_position < oldPosition) {
        std::cout<< "You passed from GO. Get 200\n";
        m_players[m_currentPlayer].m_balance += 200;
    }
}

//--------------------------------------------------------------------------------
void PlayerManager::setPosition(unsigned int i_position)
{
    m_players[m_currentPlayer].m_position=i_position;
}

//--------------------------------------------------------------------------------
unsigned int PlayerManager::getCurrentPlayer()const
{
    return m_currentPlayer;
}
```

```
//-------------------------------------------------------------------------------
unsigned int PlayerManager::getMoneyFromEachPlayer(unsigned int i_amount)
{
    unsigned int money=0;
    for(unsigned int i=0; i<m_players.size();++i)
    {
        if(m_players[i].takeBalance(i_amount))
        {
            money+=i_amount;
        }
        else
        {
            std::cout << m_players[i].m_name <<
                      " does not have enough money to pay " << i_amount <<"\n";
        }
    }
    return money;
}

//-------------------------------------------------------------------------------
void PlayerManager::setJailed(bool jailed)
{
    m_players[m_currentPlayer].m_isJailed = jailed;
    if (jailed) {
        m_players[m_currentPlayer].m_isJailed = 4;
        m_players[m_currentPlayer].m_position = 10;
    }
}

//-------------------------------------------------------------------------
PlayerManager::~PlayerManager()
{}
```

Listing 43: Property.hpp

```
#ifndef PROPERTY_H
#define PROPERTY_H

#include "Tile.hpp"

class Property: public Tile
{
public:
    //-------------------------------------------------------------------------------
    /// @brief default constructor
    /// @param[in] i_name: the name of the tile
    /// param[in] i_price: the price of the property
    //-------------------------------------------------------------------------------
    Property(const std::string &i_name, double i_price);
    //-------------------------------------------------------------------------------
    /// @brief method that prints all the information about the Property
    //-------------------------------------------------------------------------------
    void print()const;
    //-------------------------------------------------------------------------------
    /// @brief method that resets all the values of the Tile
    //-------------------------------------------------------------------------------
    void reset();
    //-------------------------------------------------------------------------------
    /// @brief method that returns the owner of the property
    //-------------------------------------------------------------------------------
    virtual unsigned int getOwner()const=0;
    //-------------------------------------------------------------------------------
    /// @brief method that does the action =p
    /// @param[in] i_player the player that have reached that specific tile
    //-------------------------------------------------------------------------------
    virtual void action(PlayerManager &i_players);
    //-------------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------------
```

```
    virtual ~Property();

private:
    //-------------------------------------------------------------------------
    /// @brief method that resets the values of any extra members of property
    //-------------------------------------------------------------------------
    virtual void resetExtras()=0;
    //-------------------------------------------------------------------------
    /// @brief method that prints any extra a property may have, ie: houses
    //-------------------------------------------------------------------------
    virtual void printExtras()const=0;
    //-------------------------------------------------------------------------
    /// @brief method that the current player pays rent to the owner
    //-------------------------------------------------------------------------
    virtual void payRent(PlayerManager &i_players)=0;

protected:
    //-------------------------------------------------------------------------
    /// @brief method to buy property
    //-------------------------------------------------------------------------
    void buyProperty(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief the owner of the the property, equals to -1 if it is not owned
    //-------------------------------------------------------------------------
    int m_owner;
    //-------------------------------------------------------------------------
    /// @brief the price of the property
    //-------------------------------------------------------------------------
    const int m_price;
    //-------------------------------------------------------------------------
    /// @brief indicated whether the property is morgated or not,1 for morgated
    //-------------------------------------------------------------------------
    bool m_isPropertyMortgaged;
    //-------------------------------------------------------------------------


};

#endif // PROPERTY_H
```

Listing 44: Property.cpp

```
#include "Property.hpp"

//-----------------------------------------------------------------------------
Property::Property(
        const std::string &i_name,
        double i_price
        ):Tile(i_name),
    m_owner(-1),
    m_price(i_price),
    m_isPropertyMortgaged(0)
{
}

//-----------------------------------------------------------------------------
void Property::print()const
{
    std::cout << "-------------------------------------------------\n";
    std::cout << "Tile's name: "    << m_name    << "\n"
              << "Tiles's colour: " << m_colour  << "\n"
              << "Price:       "    << m_price   << "\n";
    if(m_isPropertyMortgaged)
    {
        std::cout << "Mortgaged:   yes\n\n";
    }
    else
    {
        std::cout << "Mortgaged:   no\n";
    }
```

```
    printExtras();
}

//------------------------------------------------------------------------------
void Property::buyProperty(PlayerManager &i_players)
{
    char response;
    std::cout << "Would you like to buy this property? (y/n): ";
    std::cin >> response;
    while ((response != 'y') && (response != 'n')){
            std::cout << std::endl << "Please type 'y' or 'n': ";
            std::cin >> response;
            std::cout << response;
    }

    if (response == 'y') {
        if(i_players.takeBalance(m_price))
        {
            m_owner = i_players.getCurrentPlayer();
        }
        else
        {
            std::cout<< "Player does not have enough money!\n";
        }
    }
}

//------------------------------------------------------------------------------
void Property::reset()
{
    m_owner = -1;
    m_isPropertyMortgaged = 0;
    this->resetExtras();
}

//------------------------------------------------------------------------------
void Property::action(PlayerManager &i_players)
{
    if (m_owner == -1)
    {
        this->buyProperty(i_players);
    }
    else
    {
        if(m_owner!=(int)i_players.getCurrentPlayer())
        {
            this->payRent(i_players);
        }
    }
}


//------------------------------------------------------------------------
Property::~Property()
{}
```

Listing 45: Station.hpp

```
#ifndef STATION_H
#define STATION_H

#include "Property.hpp"
#include "Player.hpp"
#include <vector>

class Station : public Property
{
public:
    //------------------------------------------------------------------------
    /// @bried default constructor
```

```
    /// @param[in] i_name: the name of the tile
    /// @param[in] i_price: the price of the property
    //-------------------------------------------------------------------------
    Station(const std::string &i_name,
            double i_price,
            const std::vector<unsigned int> &i_rentPrices);
    //-------------------------------------------------------------------------
    /// @brief method that returns the owner of the property
    //-------------------------------------------------------------------------
    unsigned int getOwner()const;
    //-------------------------------------------------------------------------
    /// @brief method that prints all the information about the Station
    //-------------------------------------------------------------------------
    void printExtras()const;
    //-------------------------------------------------------------------------
    /// @brief method that resets the values of a property
    ///        in case the game is reset
    //-------------------------------------------------------------------------
    void resetExtras();
    //-------------------------------------------------------------------------
    /// @brief method that does the action =p
    /// @param[in] i_player the player that have reached that specific tile
    //-------------------------------------------------------------------------
    void payRent(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~Station();

private:
    std::vector<double> m_rentPrices;
};

#endif // STATION_H
```

Listing 46: Station.cpp

```
#include "Station.hpp"
#include <vector>

//-------------------------------------------------------------------------
Station::Station(
        const std::string &i_name,
        double i_price,
        const std::vector<unsigned int> &i_rentPrices
        ):Property(i_name,i_price)
{
    m_rentPrices.resize(4);
    for(unsigned int i=0; i<4; ++i)
    {
        m_rentPrices[i] = i_rentPrices[i];
    }
}

//-------------------------------------------------------------------------
void Station::printExtras()const
{
    std::cout << "RENT PRICES: ";
    for(unsigned int i=0; i<m_rentPrices.size(); ++i)
    {
        std::cout << m_rentPrices[i] << " ";
    }
    std::cout << "\n-------------------------------------------------\n";
    std::cout << "\n";
}

//-------------------------------------------------------------------------
void Station::resetExtras()
{
```

```
}

//------------------------------------------------------------------------
unsigned int Station::getOwner()const
{
    return m_owner;
}

//------------------------------------------------------------------------
void Station::payRent(
        PlayerManager & i_players
        )
{
    unsigned int rent = m_rentPrices[0];
    std::cout << "You have to pay " << rent << " for Rent\n";
    if(i_players.takeBalance(rent))
    {
        i_players.addBalance(rent,m_owner);
    }
    else
    {
        std::cout << "Player do not have enough money to pay";
        rent = i_players.withdrawGame();
        i_players.addBalance(rent);
    }
}

//------------------------------------------------------------------------
Station::~Station()
{}
```

Listing 47: Tile.hpp

```
#ifndef INCLUDETILE_H
#define INCLUDETILE_H

#include <iostream>
#include <vector>
#include "PlayerManager.hpp"

//------------------------------------------------------------------------------
class Tile
{
public:
    //--------------------------------------------------------------------------
    /// @brief default constructor
    //--------------------------------------------------------------------------
    Tile(const std::string &i_name);
    //--------------------------------------------------------------------------
    /// @brief method that does the action =p
    /// @param[in] i_player the player that have reached that specific tile
    //--------------------------------------------------------------------------
    virtual void action(PlayerManager &i_players)=0;
    //--------------------------------------------------------------------------
    /// @brief method that sets the colour of the tile
    //--------------------------------------------------------------------------
    void setColour(const std::string &i_colour);
    //--------------------------------------------------------------------------
    /// @brief method that gets the colour of the tile
    //--------------------------------------------------------------------------
    const std::string &getColour()const;
    //--------------------------------------------------------------------------
    /// @brief method that resets all its values to the default ones
    //--------------------------------------------------------------------------
    virtual void reset()=0;
    //--------------------------------------------------------------------------
    /// @brief method that returns the owner of a property
    //--------------------------------------------------------------------------
    virtual unsigned int getOwner()const;
    //--------------------------------------------------------------------------
```

```
    /// @brief method that returns house price
    //-------------------------------------------------------------------------
    virtual unsigned int getHousePrice()const;
    //-------------------------------------------------------------------------
    /// @brief returns the number of houses a property has
    //-------------------------------------------------------------------------
    virtual unsigned int getNumOfHouses()const;
    //-------------------------------------------------------------------------
    /// @brief method that builds a house to a property
    //-------------------------------------------------------------------------
    virtual bool buildHouse();
    //-------------------------------------------------------------------------
    /// @brief method that prints all the information about the tile
    //-------------------------------------------------------------------------
    virtual void print()const;
    //-------------------------------------------------------------------------
    /// @brief method that returns the name of the property
    //-------------------------------------------------------------------------
    const std::string &getName()const;
    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    virtual ~Tile();

protected:
    //-------------------------------------------------------------------------
    /// @brief the colour of the tile
    //-------------------------------------------------------------------------
    std::string m_colour;
    //-------------------------------------------------------------------------
    /// @brief the name of the tile i.e Piccadilly, Jail and Chance
    //-------------------------------------------------------------------------
    std::string m_name;
};

#endif // INCLUDETILE_H
```

Listing 48: Tile.cpp

```
#include "Tile.hpp"

//-------------------------------------------------------------------------
Tile::Tile(const std::string &i_name):m_name(i_name)
{
}

//-------------------------------------------------------------------------
const std::string& Tile::getName()const
{
    return m_name;
}

//-------------------------------------------------------------------------
void Tile::print()const
{
    std::cout << "-----------------------------------------------\n";
    std::cout << "Tile's name: " << m_name << "\n\n";
}

//-------------------------------------------------------------------------
unsigned int Tile::getNumOfHouses()const
{
    return 0;
}

//-------------------------------------------------------------------------
void Tile::setColour(const std::string &i_colour)
{
    m_colour = i_colour;
}
```

```
//-----------------------------------------------------------------------------
bool Tile::buildHouse()
{
    return true;
}

//-----------------------------------------------------------------------------
unsigned int Tile::getHousePrice()const
{
    return 0;
}

//-----------------------------------------------------------------------------
unsigned int Tile::getOwner()const
{
    return 0;
}

//-----------------------------------------------------------------------------
void Tile::reset()
{
}

//-----------------------------------------------------------------------------
Tile::~Tile()
{
}
```

Listing 49: Utility.hpp

```
#ifndef UTILITY_H
#define UTILITY_H
#include "Property.hpp"

class Utility : public Property
{
public:
    //-------------------------------------------------------------------------
    /// @brief default constructor
    //-------------------------------------------------------------------------
    Utility(
            const std::string &i_name,
            double i_price,
            const std::vector<unsigned int> &i_rentPrices
            );
    //-------------------------------------------------------------------------
    /// @brief method that returns the owner of the property
    //-------------------------------------------------------------------------
    unsigned int getOwner()const;
    //-------------------------------------------------------------------------
    /// @brief method that does the action =p
    /// @param[in] i_player the player that have reached that specific tile
    //-------------------------------------------------------------------------
    void payRent(PlayerManager &i_players);
    //-------------------------------------------------------------------------
    /// @brief method that resets all its values to the default ones
    //-------------------------------------------------------------------------
    void resetExtras();
    //-------------------------------------------------------------------------
    /// @brief method that prints all the information about the properties
    //-------------------------------------------------------------------------
    void printExtras()const;

    //-------------------------------------------------------------------------
    /// @brief default destructor
    //-------------------------------------------------------------------------
    ~Utility();

private:
```

```
    //-----------------------------------------------------------------------------
    /// @brief how many times you pay the total of the dices
    //-----------------------------------------------------------------------------
    std::vector<unsigned int> m_rentPrices;
};

#endif // UTILITY_H
```

Listing 50: Utility.cpp

```
#include "Utility.hpp"
#include "Dice.hpp"

//-----------------------------------------------------------------------------
Utility::Utility(
        const std::string &i_name,
        double i_price,
        const std::vector<unsigned int> &i_rentPrices
        ):Property(i_name,i_price)
{
    m_rentPrices.resize(2);
    for(unsigned int i=0; i<2; ++i)
    {
        m_rentPrices[0] = i_rentPrices[0];
        m_rentPrices[1] = i_rentPrices[1];
    }
}

//-----------------------------------------------------------------------------
void Utility::payRent(
        PlayerManager & i_players
        )
{
    Dice dice;
    dice.roll();
    const unsigned int amountTobePaid = dice.getTotal() *m_rentPrices[0];
    if(i_players.takeBalance(amountTobePaid))
    {
        std::cout << "You have paid " << amountTobePaid << " rent.\n";
        i_players.addBalance(amountTobePaid,m_owner);
    }
    else
    {
        std::cout << "You do not have enough money to pay!\n";
        i_players.withdrawGame();
    }
}

//-----------------------------------------------------------------------------
unsigned int Utility::getOwner()const
{
    return m_owner;
}

//-----------------------------------------------------------------------------
void Utility::printExtras()const
{
    std::cout << "------------------------------------------------\n";
}

//-----------------------------------------------------------------------------
void Utility::resetExtras()
{
}

//-----------------------------------------------------------------------------
Utility::~Utility()
{}
```